We connect networks -

we connect people.

Metainfo:

Current version released: Tübingen Jan 24, 2022

For dev team communications join https://matrix.to/#/#iconet-foundation:matrix.org

For any other questions or info contact via mail: steffen@iconet-foundation.org

More info about the project: https://www.iconet-foundation.org

Talk at rc3(German): https://media.ccc.de/v/rc3-2021-chaoszone-454-solving-social-ne

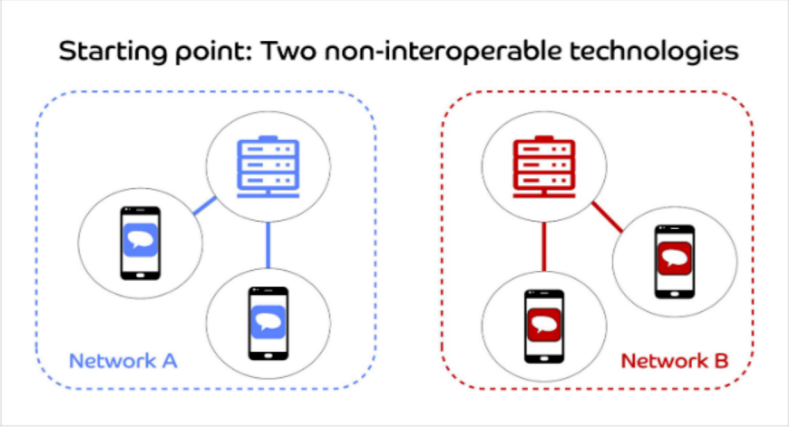
Public dev-updates via mastodon: @snac_steffen@mastodon.social
First development planing meeting on tue Jan 25. Reinforcement for the team is welcome!

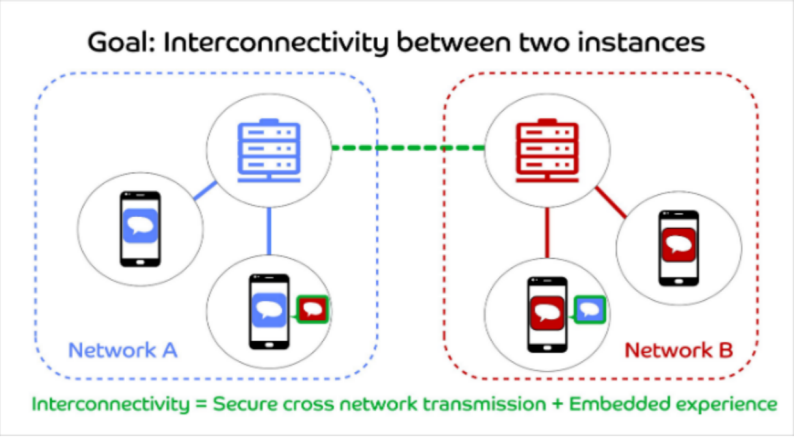
Softwareproject: iconet prototype

This documentation is intended to describe the steps necessary to complete the proof-of-concept development of interconnective social networks. The goal of this development is to modify / extend two originally incompatible technologies so that their communication functionalities can then be used by the application of the respective other technology.

This development project is detached from the actual protocol definition process. In case you want to participate there or have any questions, feel free to get in touch with us.

Overview:





At the end of development, two conditions should be achieved.

- two clients of the respective other technology can securely exchange data with each other in various ways.
- the data should be displayed within their own application via embedded-experience in the format of the other technology, interactions with the data should be possible according to the full range of functions of the other technology.

In the following, we will describe the tools used to acchieve these goals.

Part 1: Transmission tools

The transmission section describes the tools required for secure data exchange accross different systems. At first, the content and format of packages is irrelevant. We'll consider two use cases to initiate transmission: Sending data (push) and requesting data (pull). In order for these processes between independent systems to run securely, they require common addresses, authentication procedures, and interfaces for the push/pull requests. These will now be examined in more detail.

Adresses:

Adresses have two tasks:

i. Global-identifier: External services must be able to recognize from an address on which server the recipient of the package has his inbox.
ii. Local identifier: The host of the mailboxes must be able to recognize from the address to which specific mailbox the package has to be delivered. This part can either be human readable and contain e.g. information about the identity of the recipient (e.g. steffen@...) or be anonymized - also called opaque - to the outside (e.g. xu3Lm@...). However, the recipient mailbox must then still be able to assign them to the correct mailbox.

Example: In an e-mail address, the local-part is separated from the global-part by an @.

In an URL the global-part is followed by the local-part after an /.

Authentication:

Authentication includes several tasks:

i. Encryption/decryption: Sender and receiver have common mechanisms to ensure that the contents of a package cannot be read during transport to the receiver. In addition, it must be ensured that the receiver can recover the contents.
ii. Signature: A procedure to sign and verify the signator of packages. The signature of a package assures the recipient that it originates from the signer and has not been modified.
iii. Access permissions: Mechanisms to decide whether or not a user should be granted access to restricted content when requested.

Example: With asymmetric encryption methods, such as the RSA method, these requirements can be met with private and public key pairs.

push&pull requests:

Within the infrastructure, interfaces need to be available, to which external systems can deliver and request packages. The syntax of these requests must be fixed.

i. push process (sending): Packages are delivered and accepted to a host server according to the global part of the destination address on a package. The host places the package in the corresponding user-inbox based on the local-part of the destination address and forwards it to the recipient on request. There can be different packages to be send, for example: notification packages, content packages, invitations to static channels (like groups or networks)

ii. pull process (request): External systems can use an interface to request packages that their own users have made available to them. Here, it must be checked whether the request instance is authorized to do so, then the corresponding package must be delivered. There can be different requests, for example: Requesting certain package by id, requesting an overview of available packages / the users profile visible to the requestor, requesting packages through a keyword-search

Note: The implementation and processing of these requests takes place in the internal content management of the system and must be solved individually.

Part 2: Embedded experience tools

This section describes the tools needed to ensure that formats and interactions are correctly represented in the recipient's system according to the functionality of the sending system. This includes a standardized markup language, a procedure to dynamically include content in format templates, and an interface through which interactions are exchanged and managed.

Markup-language:

A markup language is choosen that is suitable for transferring formats and outlines for communication content as well as their context. The context refers to additional information that is to be displayed next to a content (e.g., display name of the author, previous interactions, ratings, ...) as well as interaction offers of the communication format (e.g., text field for commenting, quick reaction buttons, further links, ...). In the architecture of the sender technology, there must be a functionality module that provides the corresponding formats on request, and the client of the receiver needs the competence to represent formats based on this markup files.

Examples: HTML, XML, Json, ...

Data/format integration:

The architecture will enable data to be transported independently of its presentation formats. In this way, sensitive data can be exchanged directly between communication partners and an external functionality provider can provide formats and functionality independently without having access to the content itself. However, this requires mechanisms that dynamically insert the content into the correct position of the format template in the recipient's client.

Interactions:

In addition to the presentation of the content, the recipient can also be presented with various interaction options (e.g. a comment field, quick reaction buttons, further links, ...). If these active elements are used, this interaction is forwarded to the functionality module of the initiator system who processes the interaction.

These are the basic elements that we believe are needed to provide user-side compatibility between otherwise non-interoperable systems. In the prototype development, we intend to limit ourselves to these. In the actual creation of interconnective systems, additional elements will be important as well, e.g.,

notifications with previews, trust and safe spaces for secure content, anti-spam / anti-bot filters, DNS and proxy infrastructure, etc.

Where and how are these tools implemented?

These transmission tools are used differently by the sender and receiver system depending on the communication case.

> Example: The sender system encrypts a package and "labels" it with the recipient's address. The receiver system uses the address to assign the package and decrypts the package

In which instances the various elements are applied by the respective system can vary. For example, in the classic server-client model within a system, many transmission tasks are handled by the server, but encryption and decryption can be performed in the client to ensure end-to-end encryption. Trade-offs in the extent to which other personal data, such as the user's address book, is co-managed by the server system, or should be available exclusively on the user's end devices, may be decided differently from system to system. Theoretically, other architectures can also be implemented, e.g. in a peer-to-peer model the responsibilities of the server must be taken over by a client.

Process:

Having described what the goal of this software project is, let now describe the process of development.

Additional notes:

i. This is an open development and new developers are invited to join at any time. This requires that development progress is documented in such a way that it can be quickly understood by other developers. In regular meetings we discuss and organize ourselves in the developer team. For communication we have a room via matrix (https://matrix.to/#/#iconet-foundation:matrix.org). For any further questions, Martin (martin@iconet-foundation.org) and Steffen (steffen@iconet-foundation.org) are always available.
ii. There are various possible solutions for the various elements of interconnectivity, e.g. address formats could be specified differently and there are various options for the choice of markup language. In this development pragmatic decisions are to be made, other open, already existing frameworks may be used and developer preferences may play a role. In the definition of the protocol, on the other hand, it is then a matter of choosing the best option according to various criteria. However, this is a process that is detached from this prototype development.
iii. In this development, building blocks are integrated by us into another running system. Therefore, it is important that any integrations are well documented and can be easily found, so that at the end of the development there is a clear overview of all changes and developments made. Only in this way can results also be transferred to other systems.
iv. This documentation is only for overview and planning of the project. Problems that arise during development as well as circumstances that behave differently than planned here will be solved together as a team. Some development steps are intentionally simplified to be accessible to developers with less experience in communication technology development.
v. The processes described below describe only the functioning parts of the communication for better readability. In addition, the various servers and clients should always report comprehensive error messages if something does not work (e.g.: unknown address, access denied, wrong syntax, etc.).

Development plan:

Part 1: Secure cross network transmission

Step 1: Choosing an initial communication system into which we want to integrate our functionality first.

A concrete instance of a working open system must be chosen, e.g. email, ActivityPub, XMPP, matrix.org, or others. This decision will be made in the initial team-meeting on tuesday 25th January. This chosen technology will be referred to as net_A in the following.

Step 2: Setting up a local test and development environment

A client and a server should be able to be started locally from net_A, which then have the full communication functionality of net_A. Changes made in the code should become visible in the local environment. This test and development environment can be synchronized for the entire team via Git.

Step 3: Familiarization with the internal functionality of net_A.

Special attention is paid to the question which of the iconet core functionalities described above are already available in net_A, at which points an extension is appropriate and which functionalities have to be generated completely new.

Step 4: Integration of the transmission tools into net_A, initially without encryption and authentication.

First, net_A is extended by the transmission tools described above in a way, that data can be transmitted between two instances of net_A through our tools. In the process, the required elements, such as the address format, the packet syntax, the request format, etc., are continuously developed, please document any decisions made.

Intermediate steps:

a) Duplicate the server of net_A. For better overview, we recommend first setting up two instances of the same server and using one for sending and the other for receiving. (This step is optional, both sending and receiving can be integrated directly into the same server and the instance called twice for testing).

b) Duplicate the client of net_A. For better overciew, we recommend first setting up two instances of the same client and using one for sending and the other for receiving. (This step is optional, both sending and receiving can be directly integrated into the same client and the instance called twice for testing).

c) Initialization of test cases and data with test packages, test users with test addresses and inboxes.

d) Implementation of push(sending) functionality

i. The sending client can send off test packets to the sending server with the receivers address.

ii.The sending server accepts packets from its client.

iii. The sending server reads the receiver's server name from the address and forwards the packet to it (in this case, to our test receiving server).

iv. The receiving server waits for incoming packets. When the packet arrives, it checks whether, based on the local part of the address on the packet, it can assign it to one of its inboxes and then places it there.

v. The receiving client retrieves the content of his inbox from its receiving server and displays which test packets have arrived.

vi. A test case is created for the entire process where a packet is send from one client via the two servers to the other client.

e) Implementation of the pull functionality (requesting)

i. The sending client can pass test packets to his outbox in the sending server.

ii. The sending server accepts the packets from its client and places them in the outbox.

iii. The sending server waits for external pull requests and, upon receiving one, checks whether the requested object is in its outbox. If so, it sends it to the requester.

iv. The receiving client can initiate addressed pull requests and forward them to its receiving server.

v. The receiving server can accept pull requests from its receiving client and request them from the sending server. The sending server forwards the response directly to its client.

vi. The receiving client can accept the forwarded packets and display them.

vii. A test case is created for the entire process, where a packet can first be uploaded in one client and then requested from the other client through the two servers.

Step 5: Include encryption and authentication.

The test systems are now extended to include encryption and authentication mechanisms. For this purpose, it is helpful to familiarize oneself with implementations on asymmetric encryption and signature and, if necessary, to understand their application in more detail first in a separate test environment.

Intermediate steps:

a) Extension of test data: The test users receive their own private keys as well as the respective other public keys.

b) Implementation in the sending client to encrypt the contents of an outgoing test packet with the receivers's public key before sending.

c) Implementation in the receiving client to decrypt the contents of an incoming packet using its own private key.

d) Extension of the pull process: requests are now checked for authentication beforehand.

i. The sending client appends a list of public keys to the transfer of the test packet to the sending server's outbox. (Only requests are signed with a private keys matching the public keys in this list are allowed access to the uploaded test packet.)

ii. The sending server stores this key list with the test packet. If the sending server receives a pull request, it checks its signature. Only if it matches the stored keys, the test packet is sent.

iii. The receiver client signs outgoing pull requests with its private key before sending them.

e) The sending client not only encrypts test packets, but also signs them with its private key before sending them.

f) The recipient client not only decrypts test packets, but also checks them for their signature using the contacts' known public keys.

g) If necessary, the existing test cases and outputs of the clients are extended to reveal the encryption and authentication capabilities of the system.

Step 6: The transmitting architecture and the receiving architecture are combined.

Each system should of course be suitable for transmitting as well as for receiving. According to personal preference it can be decided in which direction the program code is integrated. Test cases are to be extended in such a way that now in both directions data can be sent. This step is omitted if the distinction in 4a) and 4b) has already been skipped.

Note: The upcoming steps are documented in less detail from this point on and serve more as an overall overview. At the time of publishing this version, what counts most is that the first steps are clear. The planning of the details and the sequence of the further steps will be done as needed.

Step 7-12: Choosing a second communication system net_B into which we integrate our transmission functionality, for which we repeat the steps 1-6.

To make the added value of interconnectivity clear, it is necessary to exchange data between two previously incompatible systems. For this purpose, another already existing functional instance of any technology is first selected, a test environment is set up and the team familiarizes them with the inner workings of net_B. Then, all the building blocks that were used in net_A in steps 4-6 are developed for net_B as well. For this purpose, the results from the previous development can be used as a basis, but the individual requirements of net_B must be taken into account, potentially a full reimplementation must be done. It is only important that the communication interfaces and syntaxes of the interconnective elements are adapted identically as done in net_A. In case some changes to interfaces need to be done, the implementations for net_A need to be adapted accordingly.

Step 13: Combining net_A and net_B

At this point, we have implementations of net_A and net_B, which have the capabilities to transmit data through the same interfaces. Now we need to create combined test cases. The communication should function in both directions, net_A as receiving system and net_B as transmitting system, as well as vice versa.

Part 2: Implementation of Embedded experience

This part of the documentation will be added to a later time. If you have any questions regarding this part now, don’t hesitate to get in touch with us!